BASEMENT

MASSACHUSETTS
INSTITUTE of TECHNOLOGY
LIBRARIES

HD28
.M414
NO.1333-
82

Dewey
MASS. INST. TECH
JAN 11 1983
LIBRARIES

A MODEL
OF
SOFTWARE PROJECT MANAGEMENT DYNAMICS

Tarek K. Abdel-Hamid
Stuart E. Madnick

May 1982

CISR WP #91

Sloan WP #1333-82

Center for Information Systems Research
Massachusetts Institute of Technology
Sloan School of Management
77 Massachusetts Avenue
Cambridge, Massachusetts, 02139

A MODEL
OF
SOFTWARE PROJECT MANAGEMENT DYNAMICS

Tarek K. Abdel-Hamid
Stuart E. Madnick

May 1982

CISR WP #91

Sloan WP #1333-82

© T. K. Abdel-Hamid S. E. Madnick 1982

This paper will be presented at the Sixth International Computer Software and Applications Conference (COMPSAC), November 8-12, 1982.

Center for Information Systems Research
Sloan School of Management
Massachusetts Institute of Technology

M.I.T. LIBRARIES
JAN 1 1 1993
RECEIVED

ABSTRACT

This paper is a study of the dynamics of software project management. In Section (I) we argue for the attractiveness of using computer modeling techniques in the study and design of complex social systems. In particular, we advocate the use of the modeling, simulation, and analysis techniques of System Dynamics. In Section (II) we, then, present a System Dynamics model we developed for software project management. The model's three major subsystems: the human resource management subsystem, the planning and control subsystem, and the software production subsystem, are discussed in some detail. Finally, in Section (III), the dynamic behavior of the model is analyzed using a series of simulation runs.

Key Words and Phrases: Management of Software Development, Project Management, Simulation, Software, System Dynamics.

0745564

I. COMPUTER MODELING OF SOCIAL SYSTEMS

> People would never attempt to send a space ship to the moon without first testing the equipment by constructing prototype models and by computer simulation of the anticipated space trajectories. No company would put a new kind of household appliance or electronic computer into production without first making laboratory tests. Such models and laboratory tests do not guarantee against failure, but they do identify many weaknesses which can then be corrected before they cause full-scale disasters.
> Our social systems are far more complex and harder to understand than our technological systems. Why, then, do we not use the same approach of making models of social systems and conducting laboratory experiments on those models before we try new laws and government programs in real life? The answer is often stated that our knowledge of social systems is insufficient for constructing useful models. But what justification can there be for the apparent assumption that we do not know enough to construct models but believe we do know enough to directly design new social systems by passing laws and starting new social programs? I am suggesting that we now do know enough to make useful models of social systems. Conversely, we do not know enough to design the most effective social systems directly without first going through a model-building experimental phase. But I am confident, and substantial supporting evidence is beginning to accumulate, that the proper use of models of social systems can lead to far better systems, laws, and programs. (Forrester, 1971)

Indeed it is now possible to use the modeling, simulation, and system analysis techniques of System Dynamics to construct in the "laboratory" models of managerial systems. Such models are obviously simplifications of the actual social systems but can be far more

comprehensive than the mental models that are otherwise used.

System Dynamics is the application of feedback control systems principles and techniques to managerial and organizational problems. The first analytical techniques were first conceived within the engineering disciplines to design complex feedback control systems like automatic pilots. It became apparent, however, that these techniques could also be applied to the study of business and social systems whose behavior is caused by the feedback nature of their intrinsic structures.

The System Dynamics philosophy rests on a belief that the behavior (or time history) of an organizational entity is principally caused by its structure. The structure includes, not only the physical aspects, but more importantly the policies and traditions, both tangible and intangible, that dominate decision-making in the organizational entity. Such a structural framework contains sources of amplification, time lags, and information feedback similar to those found in complex engineering systems. Engineering and management systems containing these characteristics display complicated response patterns to relatively simple system or input changes. The analysis of large nonlinear systems of this sort is a major challenge to even the most experienced control systems engineer; effective and reliable redesign of such a system is still more difficult. The subtleties and complexities in the management area make these problems even more severe (Roberts, 1981). Here the structural orientation of System Dynamics provides a methodology for replacing confusion with order.

The System Dynamics approach begins with an effort to understand the system of forces that has created a problem and continues to sustain it. Relevant data are gathered usually from a variety of sources (e.g., literature, informed persons, ... etc). As soon as a rudimentary measure of understanding has been achieved, a formal model is developed. This model is initially in the form of a set of logical diagrams showing cause-and-effect relationships. As soon as feasible the visual model is translated into a mathematical version. The model is exposed to criticism, revised, expressed again and so on in an iterative process that continues as long as it proves to be useful. Just as the model is improved as a result of successive exposures to critics, a successively better understanding of the problem is achieved by the people who participate in the process.

Such an approach forces those involved in system design to make explicit, and thoroughly test the assumptions that underlie their design decisions: the nature of problems, their causes, the consequences of alternative actions, and how various human, managerial, economic, and operational factors interrelate. Weil reports that his experience has shown that this is a very valuable process ... that people are really quite surprised when it turns up things no one had thought of before, incorrect assumptions, and differences of opinion about cause and effect (Weil, 1981).

Roberts has stated that people's " ... intuition about the probable consequences of proposed policies frequently proves to be less reliable than the model's meticulous mathematical approach ... This is

not surprising as it may first appear. Management systems contain as many as 100 or more variables that are known to be related to one another in various non-linear ways. The behavior of such a system is complex far beyond the capacity of intuition. Computer simulation is one of the most effective means available for supplementing and correcting human intuition." (Roberts, 1981)

System Dynamics is suitable for addressing certain kinds of complex problems. In addition to complexity, such problems have at least two features in common. First, they are dynamic i.e., they involve quantities which change over time, and which can, therefore, be expressed in terms of graphs of variables over time. Oscillating levels of employment in an industry, a decline in a city's tax base and quality of life, and the dramatically rising pattern of health care costs are all dynamic problems (Richardson and Pugh, 1981). So too are the cost overruns, slippages in scheduled completion dates, and the variability in productivity in a software project.

"A second feature of the problems to which the System Dynamics perspective applies involves the notion of feedback ... "Most succinctly, feedback is the transmission and return of information. The emphasis, inherent in the word feedback itself is on the return." (Richardson and Pugh, 1981) A feedback system exists whenever an action taker will later be influenced by the consequences of his or her actions. The consequences may be quick and directly apparent in results produced e.g., as when the hiring of ten more programmers increases the programmers' workforce to a certain desired level, which

in turn feeds back to affect the hiring rate i.e., in this case stopping further hiring. Or the consequences may be delayed though directly apparent in results produced e.g., as when a software development manager's decision to use a particular package to estimate his/her human resource requirements affects the project's completion time and cost, which in turn influences the manager's later estimation procedure. Finally, the consequences may be both delayed and quite indirect in perceived results e.g., as when a decision to increase the software development budget leads to the hiring of higher quality managers and analysts/programmers, who may then develop improved products several years later, which may enhance the company's competitive position, in turn increasing sales and/or profits, which may then influence the decision on the software development budget. In all these cases a "closing of the loop" occurs. And a delay, whether short or long, intervenes between initial action and fedback results. Closed loops and time delays in consequences are characteristic of all feedback processes.

II. A SYSTEM DYNAMICS MODEL OF SOFTWARE PROJECT MANAGEMENT

It is apparent from the above discussion that the management of software development is a prime candidate for application of the System Dynamics method. It clearly is complex, it is dynamic, and it does exhibit feedback behavior as we showed above.

In this section we present a System Dynamics model we developed of software project management. The model was developed to be a tool in investigating the many managerial problems that seem to be plaguing software development activities in most organizations e.g., cost and schedule overruns. In particular, the present model will serve as a "skeleton" on top of which "custom" models will be built to fit the specific organizations we will be studying.

The model consists of three subsystems: (1) The Human Resource Management Subsystem (HRMS); (2) The Planning and Control Subsystem (P&CS); and (3) The Software Production Subsystem (SPS) as shown in Figure (1).

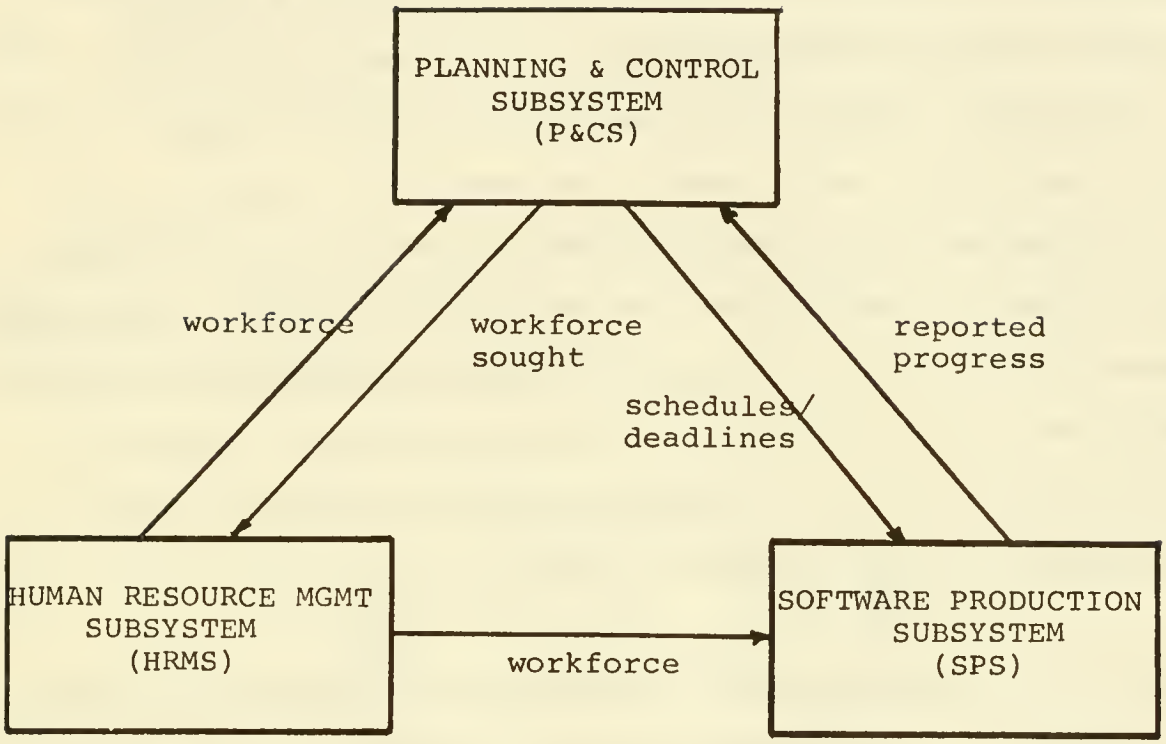


Figure (1)

The Model's Subsystems

The HRMS contains the hiring/firing and training policies of the organization. The P&CS includes the managerial activities for assessing project status, projecting future personnel needs, and adjusting project schedules if necessary. And finally, the SPS captures the software "production" activities such as designing, coding, and testing. Figure (1) also shows some of the major information connections which couple the subsystems together. For example, information on the project's progress and on the workforce level flows into the Planning and Control Subsystem (P&CS) from SPS and HRMS respectively. Such information is used to decide whether or not more people need to be hired and/or schedules adjusted. Once such decisions are made, they are forwarded (after some delay) to the appropriate subsystem(s), as shown in Figure (1).

Each of the three subsystems is discussed in some detail, below. However, because all the diagrams will be in terms of the schematic conventions used in System Dynamics, we feel it would be useful to preface the discussion with an introduction to these conventions.

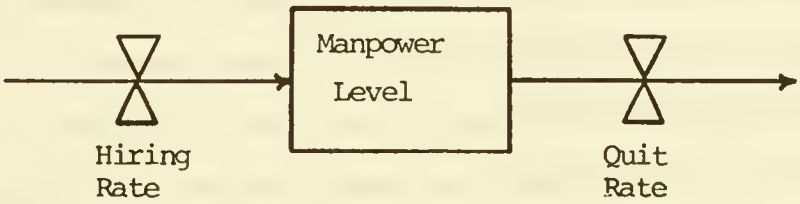
From a System Dynamics perspective all systems can be represented in terms of "level" and "rate" variables, with "auxiliary" variables used for added clarity and simplicity.

A level is an accumulation, or an integration, over time of flows or changes that come into and go out of the level. The term "level" is intended to invoke the image of the level of a liquid accumulating in a container. The system dynamicist takes the simplifying view that

feedback systems involve continuous, fluid-like processes, and the terminology reinforces that interpretation.

The flows increasing and decreasing a level are called rates. Thus, a manpower pool would be a level of people that is increased by the hiring rate and decreased by the firing and/or quit rate.

Rates and levels are represented as stylized valves and tubs, as shown below, further emphasizing the analogy between accumulation processes and the flow of a liquid.



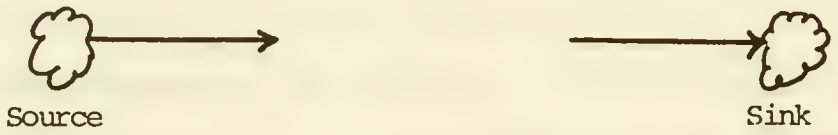
The flows that are controlled by the rates are usually diagramed differently, depending on the type of quantity involved. We will use the two types of arrow designators shown below:

Information Flows

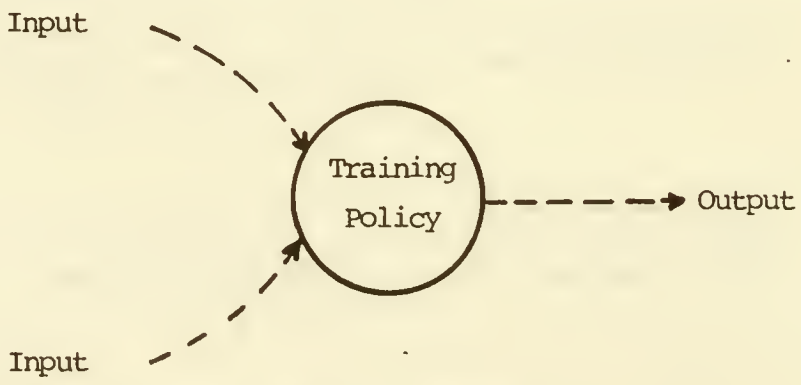
Other Flows
(e.g., People)

Flows will always, of course, originate somewhere and terminate

somewhere. Sometimes, the origin of a flow is treated as essentially limitless, or at least outside the model-builder's concern. In such a case the flow's origin is called a source. Similarly, when the destination of a flow is not of interest, it is called a sink. Both sources and sinks are shown as little "clouds."



All tangible variables are either levels or rates i.e., they are either accumulations of previous flows or are presently flowing. But there is one more type of information variable, which is called an auxiliary. Auxiliary variables are combinations of information inputs into concepts e.g., "desired workforce," or policies e.g., "training policy." Auxiliaries are represented by a circular symbol as shown below:



The Human Resource Management Subsystem:

As shown in Figure (2), the workforce is disaggregated in the model into two categories of employees: new recruits and experienced employees. In the model the work category of an employee is a key (but not the only) determinant of his/her average productivity. The average new recruit just joining the firm cannot be expected to be as effective as the employee who has been with the firm for a longer period of time. This is not due merely to the overall number of years of software development experience, since new recruits may come from other jobs with many years of software background. The process of indoctrination and orientation itself requires several months before the new employee becomes effective.

Experienced employees, on the other hand, devote some of their time training the new comers. How much time is devoted to such training is a management policy that is captured in the model by a simple parameter that can be easily altered for different simulation runs. Notice that the rate at which new recruits are assimilated into the organization is shown in Figure (2) to be affected by the training policy i.e., a high level of training leads to a high assimilation rate. There is a tradeoff, however, since a high level of training means that a high fraction of the experienced employees' effort will be siphoned off into the training function.

The hiring/transfer policy is driven by the gap between the number of employees desired and those available. The number of employees desired is determined (as shown in Figure (1)) in the Planning and

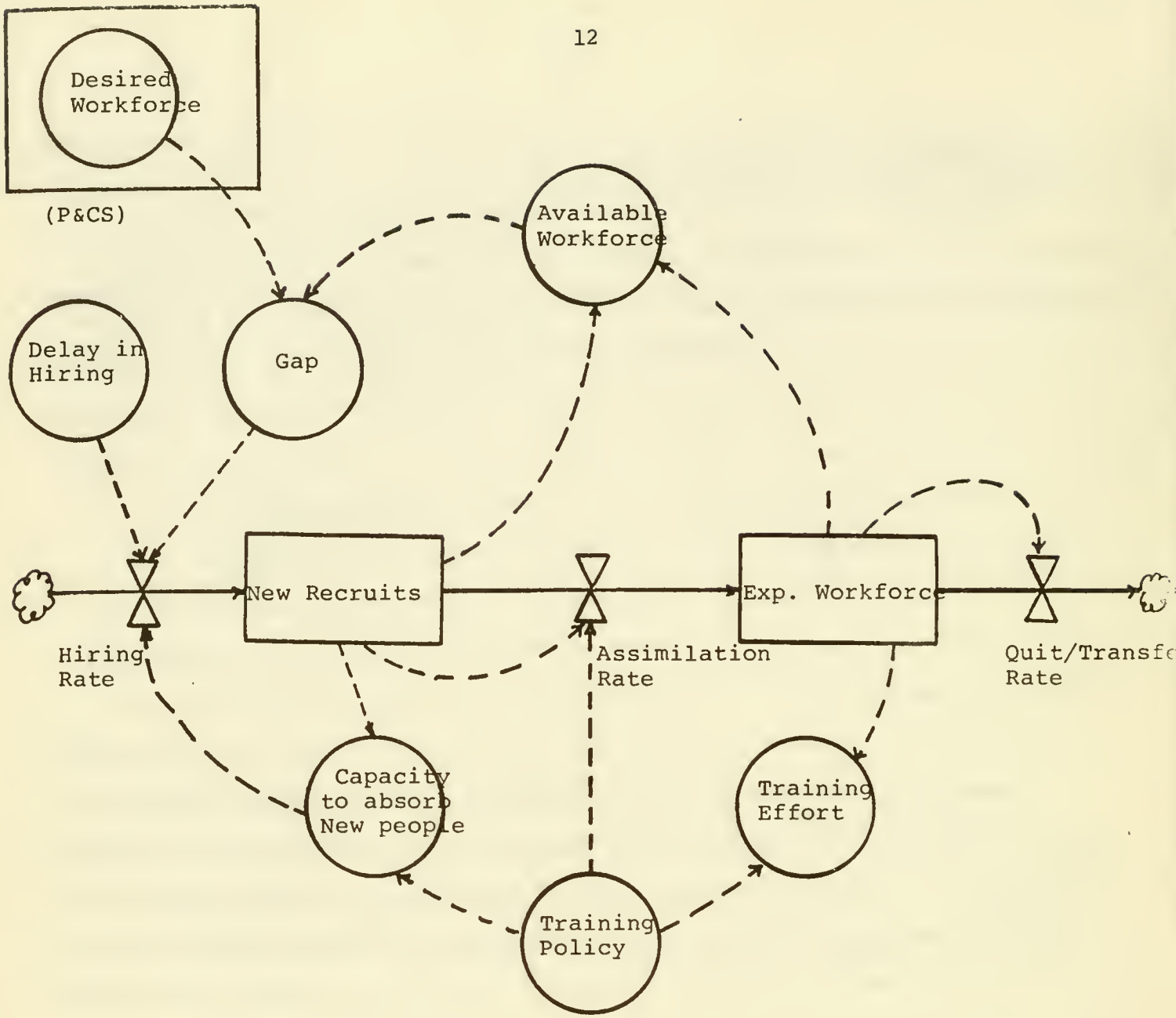


Figure (2)

The Human Resource Mgmt Subsystem (HRMS)

Control Subsystem. The rate at which the organization will hire new people is a function of three factors in the model. The workforce gap combines with the organization's delay in changing its level of employees to produce a "desired" rate of change in manpower. The delay represents the fact that only a fraction of the personnel requests shown by the gap can be processed in any single month. However, the organization restricts its hiring rate, not soley because of its own limitations in recruiting capability nor because of the limits imposed by the lack of immediate availability of qualified software professionals. In addition to these, the organization takes acount of its own ability to absorb new people into the organization, to train them, and make them an integral part of a particular team. Thus, the model incorporates the (usually implicit) policy in which the rate of hiring of new employees is restricted to that number which the organization feels its fully integrated staff can handle.

The Planning & Control Subsystem (P&CS)

As shown in Figure (3) the P&CS operates largely on the basis of progress information supplied by the Software Production Subsystem throughout the project's lifecycle. On the basis of the man-months and time perceived remaining, management can compute the number of people (believed) required to complete the project on schedule. If this indicated workforce level is higher than that of the current workforce level, one of three things can be done: hire more people, extend the schedule, or do a little of both.

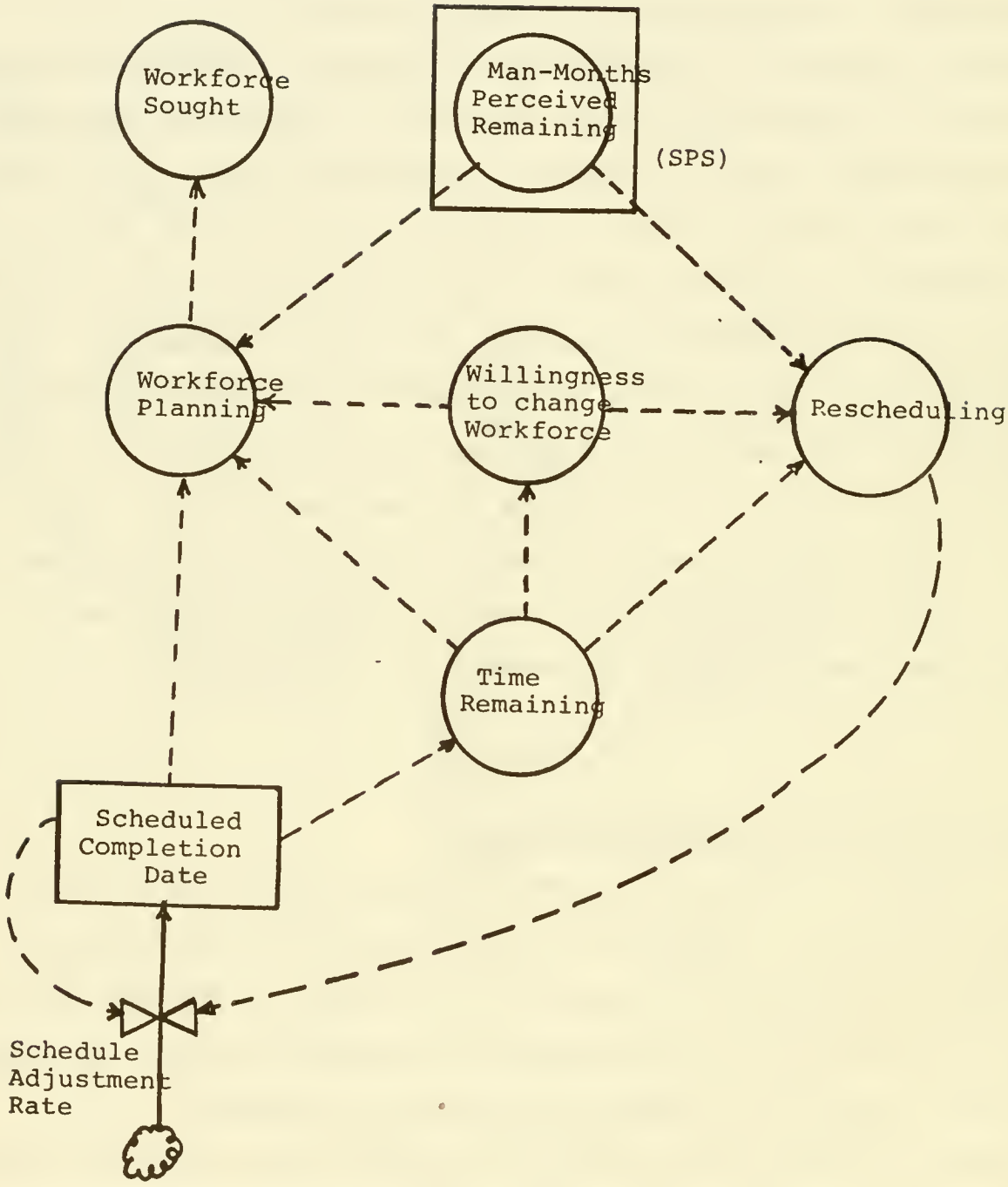


Figure (3)

The Planning & Control Subsystem (P&CS)

When deciding upon the number of employees it desires to have, management (in the model) takes into account the time perceived remaining for the project. Toward the end of the project, for example, there could be considerable reluctance to bring on new people, even though the time and effort perceived remaining imply more people are needed. It would just take too much time to acquaint new people with the mechanics of the project, integrate them into the project team, and train them in the necessary technical areas.

The Software Production Subsystem (SPS):

The SPS is depicted in Figure (4). The software projects we will model will be defined in terms of a number of "tasks." (The task unit will be defined later.) Progress in the project is made as workers perform tasks. Not all work done in the course of a large software project is errorless, however. Some fraction of it will be less than satisfactory and will be redone. Unsatisfactory work is not discovered right away, though. For some time it passes unrecognized, until the need for reworking the tasks involved shows up. Thus, at any point in time, the cumulative number of tasks perceived completed, is the cumulative number of tasks worked minus the number of tasks that are known to be flawed.

In addition to tracking progress, management also tracks effort expenditures in man-months e.g., using "time cards" or their equivalent. By comparing cumulative expenditures to cumulative progress e.g., using an "Earned Value System," trouble spots can be

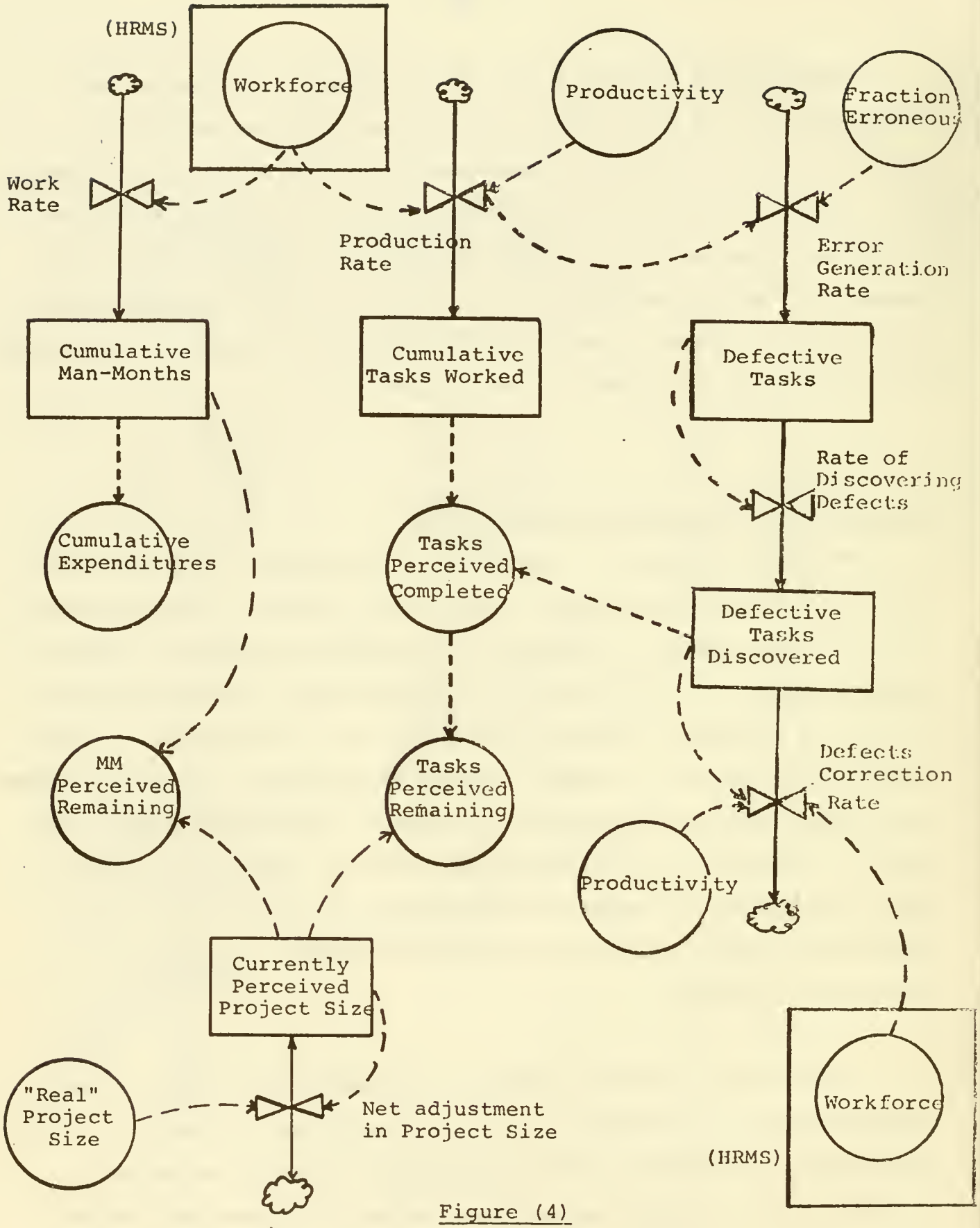


Figure (4)

The Software Production Subsystem (SPS)

identified and remedial action taken e.g., adjusting the size of the project upwards.

III. ANALYSIS OF SYSTEM BEHAVIOR

One of the attractive features of computer modeling is their high versatility ... by simply changing a few model parameters, for example, one can easily simulate a wide range of software project types. The software project to be discussed here involves the development of a 400,000 DSI system. (DSI stands for "Delivered Source Instructions." See (Boehm, 1981) for a complete definition.) All of the model's graphical output, however, will be in terms of the unit "task," where a "task" is equivalent to 400 DSI. (Our project is, thus, of size 1000 tasks.) The development period modeled begins at the beginning of the product design phase and ends at the end of the integration and test phase. We deliberately excluded the requirements definition/specification phase so we may incorporate the use of TRW's "COnstructive COst MOdel" (COCOMO). (Boehm, 1981) That is, we will assume that management will use COCOMO to estimate the effort in man-months (MM) that will be required, the total development time in months (TDEV), and the staff size (SS).

Run (1): The "Flawless" Project

In the "ideal" case, management's estimate of the project's size will be completely on target i.e., 400,000 DSI or 1,000 tasks. Using the (Basic form of the) COCOMO model, an estimate of the effort in man-months (MM) can be made as follows:

$$MM = 2.4(DSI/1000)^{1.05}$$
$$= 2.4\ (400{,}000/1000)^{1.05}$$
$$= 1295 \text{ man-months}$$

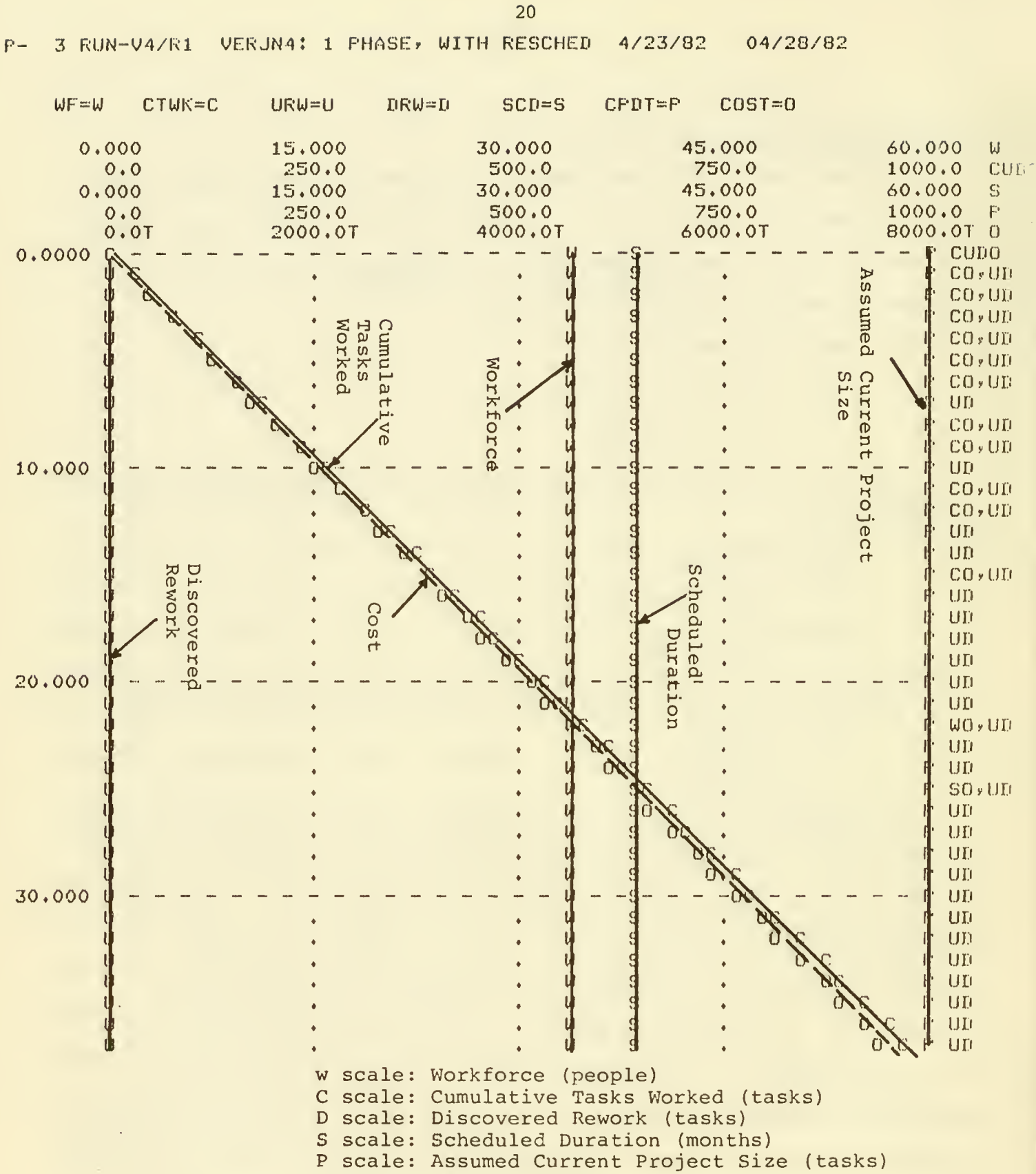
From this an estimate of the project's total development time can be calculated:

$$TDEV = 2.5(MM)^{0.38}$$
$$= 2.5(1295)^{0.38}$$
$$= 38 \text{ months}$$

Finally, the average staff size (SS) is determined as follows:

$$SS = MM/TDEV$$
$$= 34 \text{ people}$$

The dynamic behavior of the "flawless" project situation is shown in Figure (5). The project's scheduled completion duration is set to 38 months and a workforce of 34 people is assembled (e.g., during the requirements/specification phase). As can be seen, the project proceeds "smoothly" and completes on schedule at a total cost of $ 7,817,000. In calculating the total cost we are using a cost of $ 6,000/man-month. (It is assumed that a rather uniform effort is required throughout the project. This simplifying assumption will be relaxed in a later version of the model in which the software lifecycle will be explicitly divided into three phases: design, coding, and testing.)



w scale: Workforce (people)
C scale: Cumulative Tasks Worked (tasks)
D scale: Discovered Rework (tasks)
S scale: Scheduled Duration (months)
P scale: Assumed Current Project Size (tasks)
O scale: Cost ($ 1,000)

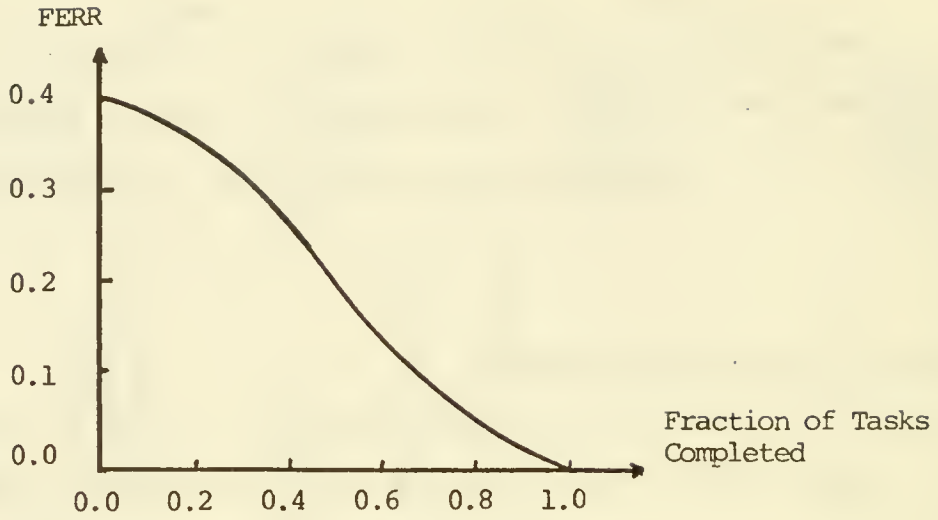
Figure (5)

Unfortunately the project behavior of Figure (5) is rarely (if ever) realized in real organizations. It will, therefore, only serve us as an "interesting" reference point for further analysis.

Run (2): Introducing Rework

Not all work done in the course of a large software project is errorless. Some fraction of the work will be less than satisfactory (e.g., inconsistent design, defective code, ... etc) and must be redone. The unsatisfactory work is not discovered right away, however. For some time it passes unrecognized, until the need for reworking the tasks involved shows up. The discovery of unsatisfactory work that needs reworking can, of course, cause major disruptions in a software project e.g., as people are diverted from new tasks to redoing old ones, and is, therefore, a significant element of the software development environment.

In the model, the generation of rework is regulated by "FERR," which is the fraction of work that is erroneous. For example, a value of FERR equal to 0.1 means that 10 percent of the tasks completed in a particular unit of time will be defective, and will thus require reworking. FERR is not modeled as a constant, however. It is a variable that changes during the life of the project. The curve below depicts the assumed relationship between FERR and the fraction of tasks completed.



The rationale for making FERR a variable in this fashion is the realization that tasks near the beginning of a large software project (e.g., design) are much different from those near the end (e.g., documentation). Near the beginning, the project is being defined, different approaches are being explored, ideas are on the drawing board ... etc. Near the end, on the other hand, the tasks look more like finishing touches, assembling documentation, typing reports, and so on. The likelihood of performing work that must be redone is thus modeled to be greater at the beginning of the project than at the end.

The behavior of the model is shown in Figure (6). At the bottom of the figure, the level of "rework" perceived (i.e., discovered) throughout the life of the project is shown. The generation of rework means of course that more effort will be required to complete the project satisfactorily. After month 10 enough rework is discovered to cause management to start hiring more people in order to meet the

P- 6 RUN- VERJN4: 1 PHASE, WITH RESCHED 4/23/82 04/28/82

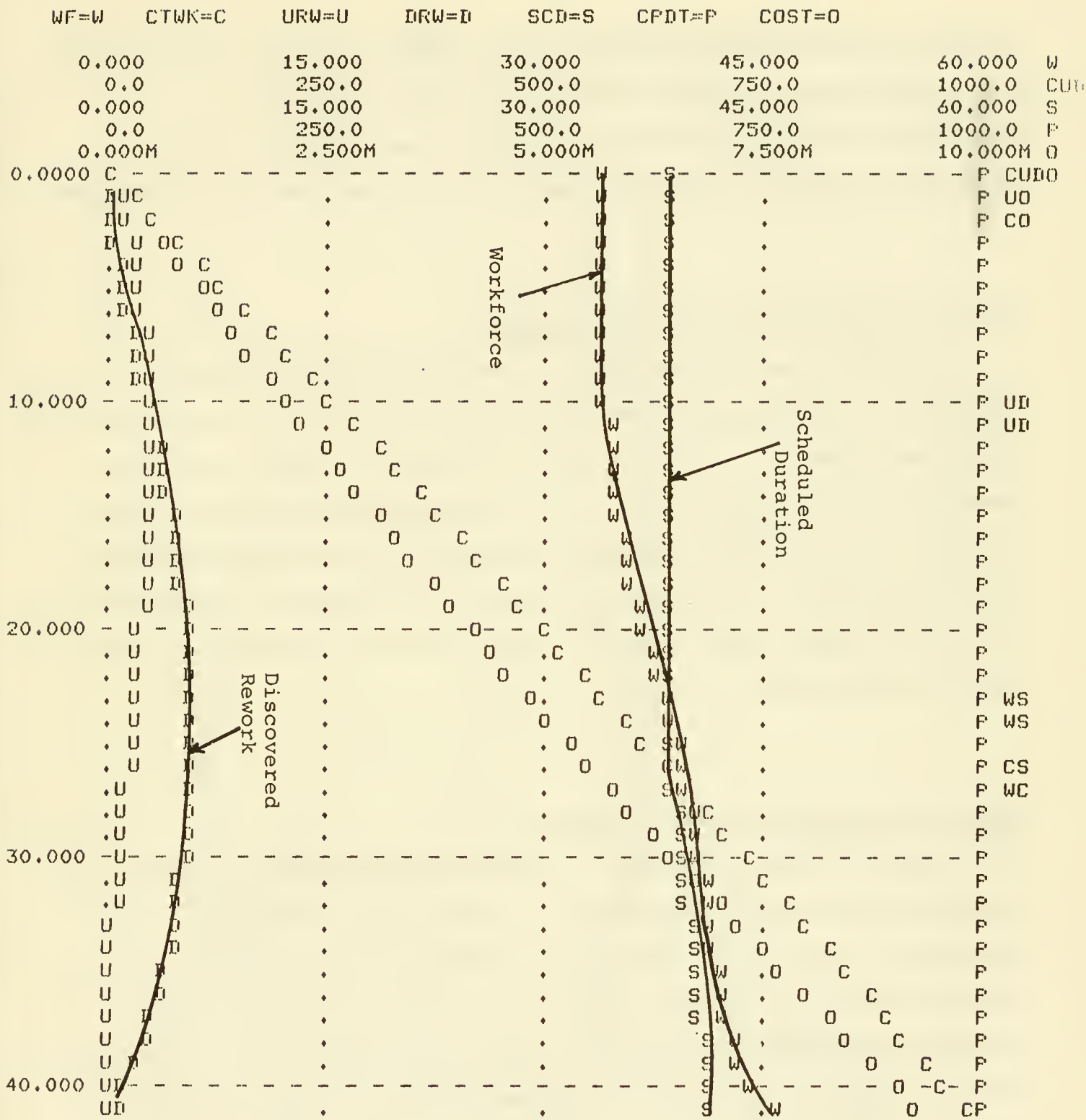


Figure (6)

initial scheduled completion date. This policy continues until month 28, when management chooses instead to extend the project's target completion date and slow down the hiring of new people. The project eventually completes after 41.5 months, at a total cost of $ 9,390,700.

Run (3): Introducing Personnel Turnover

In this run we diverge further from the "flawless" project situation. In addition to the generation of rework, we introduce the effect of personnel turnover. In the "flawless" project run it was assumed that people don't quit. (After all who would quit an "ideal" project!) For this run, though, we assume that the average employment time is 36 months. As shown in Figure (7), the project is adversely affected finishing even later, at month 46 and at the slightly higher cost of $ 9,636,000.

Run (4): Introducing Estimating Error

In the "flawless" project we assumed that management's estimate of the project's size was completely on target i.e., 1,000 tasks. Obviously, this is too optimistic an assumption. Boehm states that at the beginning of a software project " ... When we first begin to evaluate alternative concepts for a new software application, the relative range of our software cost estimates is roughly a factor of four on either the high or the low side." (Boehm, 1981)

For this run, we will assume that management initially estimates

P- 9 RUN- VERJN4: 1 PHASE, WITH RESCHED 4/23/82 04/28/82

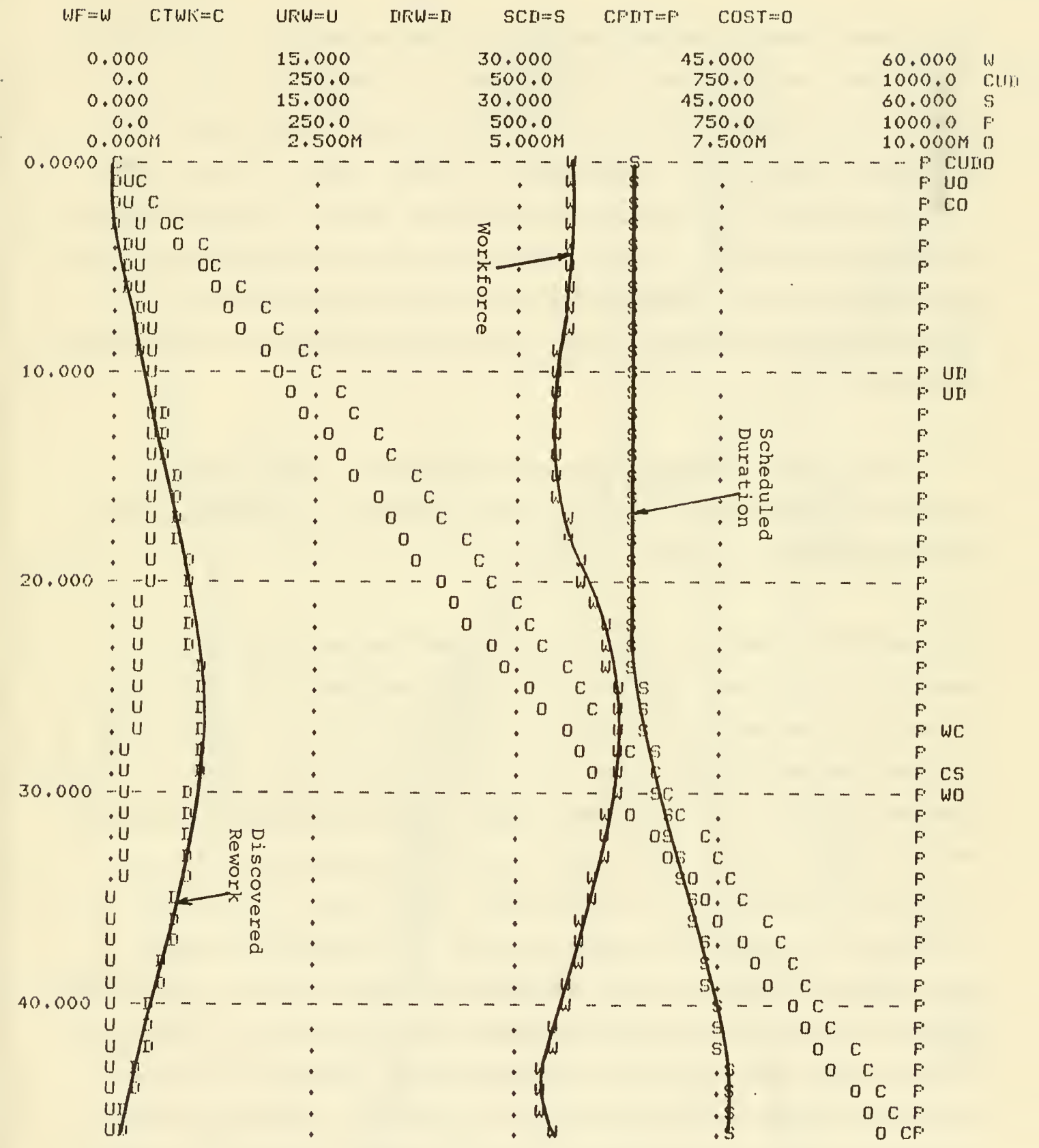


Figure (7)

the system's size to be 600,000 DSI or 1,500 tasks i.e., 50 percent higher than the real size of 400,000 DSI or 1,000 tasks. This situation creates the following interesting observation: The total development time (TDEV) produced by the COCOMO model for the (estimated) 1,500 task project is 45 months. What is interesting about this figure is that it is very close to the actual completion date (46 months) in Run (3). "Knowing" that a 45 month project duration is an attainable schedule we would have little reason to expect any schedule overruns.

What actually happens is shown in Figure (8). The project still overruns its scheduled completion date, finishing in 52 months and at a total cost of $ 9,394,000.

Notice in the figure that the "assumed current project size" starts at 1,500 tasks, but as the project progresses, and the level of uncertainty decreases, it is adjusted downwards until at about month 35 "assumed current project size" is in fact equal to 1,000 tasks ... the true project size. As management learns about the downward adjustments in the project's size, it adjusts its workforce downwards (e.g., by transferring people out of the project) to the level it perceives is sufficient to complete the project within the scheduled 45 months. What happens, though, is that unexpected problems arise e.g., a system integration test fails miserably, which will necessitate the reworking of tasks that were believed to be successfully completed. When such disruptions occur towards the end of the project, when management is reluctant to hire new employees, there is almost no other alternative

P- 12 RUN- VERJN4: 1 PHASE, WITH RESCHED 4/23/82 04/28/82

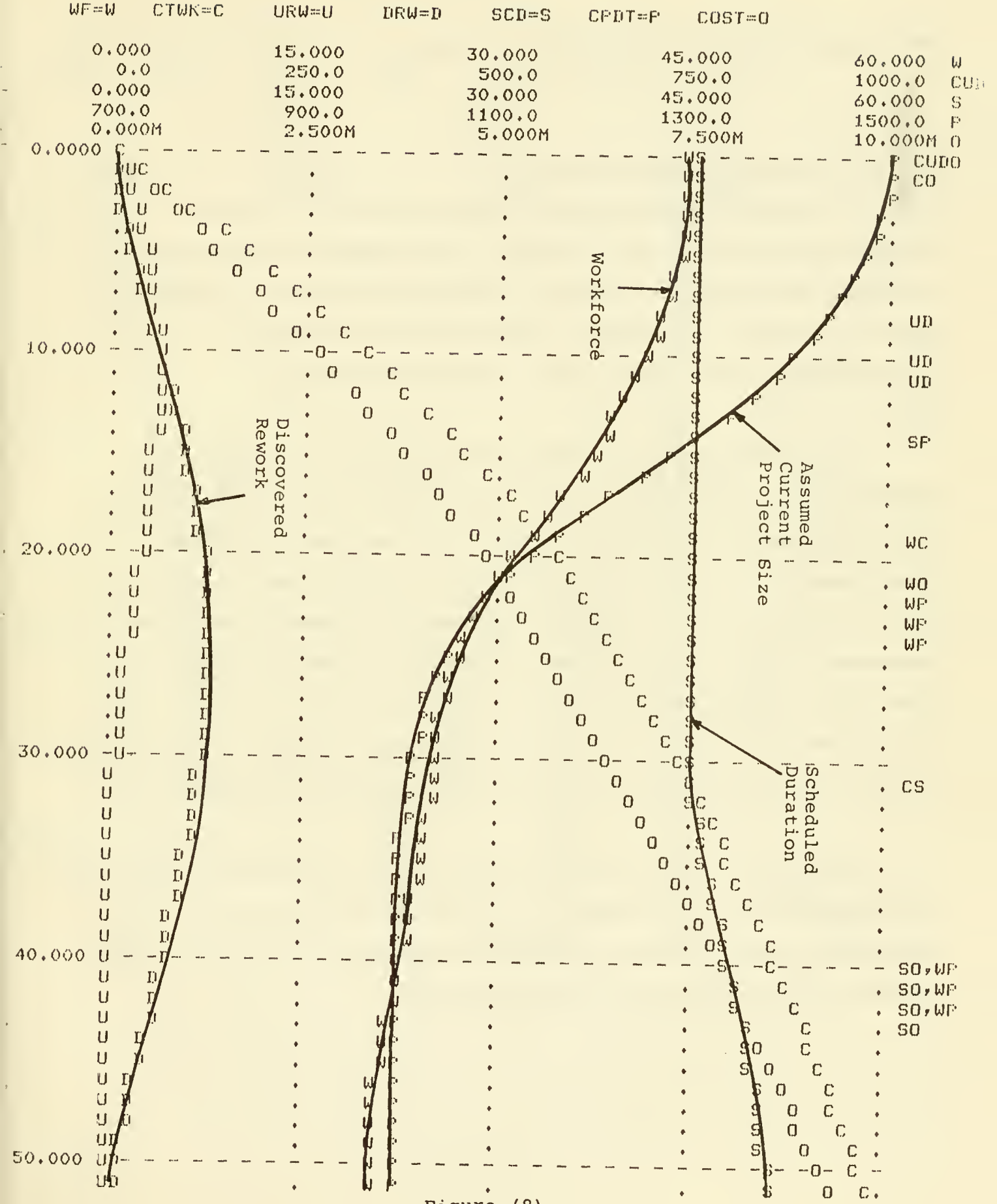


Figure (8)

but to extend the scheduled comletion date ... as shown in Figure (8).

Run (5): Fixed-Workforce Policy

In our two final runs, we will make use of the "what-if" capabilities of our computer model to investigate the impact of two (extreme) managerial policies for software development. In both runs, we will maintain our project's present level of "realism," i.e., include rework, personnel turnover, and overestimation.

In the first scenario, management places a lot of weight on maintaining a stable workforce, while, at the same time, being quite willing to immediately adjust schedules in response to project "surprises." The project's dynamic behavior under such a policy is shown in Figure (9). The project is completed ahead of schedule, consuming a total of only 36 months at a total cost of $ 9,567,000. The initial estimation error leads to a high initial workforce level (of 44 people), which is then maintained (unnecessarily) throughout the project, causing the project to be completed ahead of schedule.

Clearly the merits of such a policy cannot be adequately discerned without additional knowledge (e.g., from the firm's marketting department) that, somehow, assesses the benefits (if any) that are realized from finishing the project sooner.

P- 21 RUN- VERJN4: 1 PHASE, WITH RESCHED 4/23/82 04/28/82

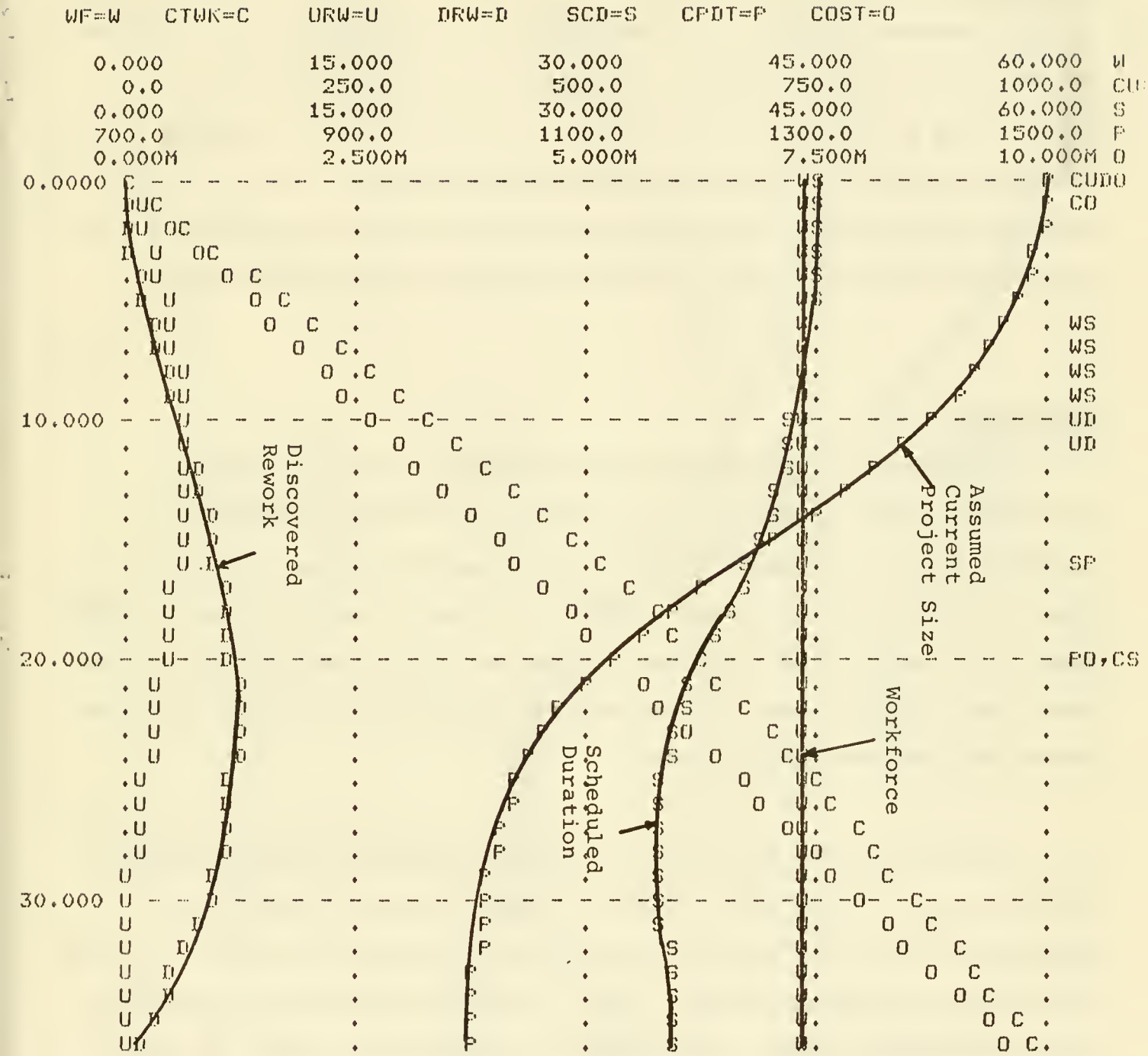


Figure (9)

Run (6): Fixed-Schedule Policy

In contrast to the above case, Figure (10) presents the situation in which the initial schedule is treated as more or less fixed. Maintaining a stable workforce has a low priority here. Indeed, towards the end of the project we see the workforce level shooting upwards as management maintains an unwavering resolve to complete the project on schedule. The total cost for this case is $ 9,505,000.

Conclusion

Our purpose in this section was to demonstrate the viability of the System Dynamics approach for modeling and studying the dynamics of software project management. We have attempted to give the reader a flavor of what System Dynamics offers in terms of capabilities to model managerial actions and policies, "deducing" their dynamic implications over time, and also the (convenience) of answering "what-if" questions about such actions and policies.

As we mentioned previously, the System Dynamics model discussed here is only a first step "skeleton" model, on top of which more detailed and better "tailored" models will eventually be built for some real organizational settings. Still , we feel the results produced by this (simplified) model, and which are summarized in Table (1) below, provide some interesting insights into the management of software projects. For example, overruns in the cost of developing software seems to be largely due to the necessity to rework tasks, either because of changes in (user) requirements or the discovery of errors.

P- 15 RUN- VERJN4: 1 PHASE, WITH RESCHED 4/23/82 04/28/82

WF=W CTWK=C URW=U DRW=D SCD=S CPDT=P COST=O

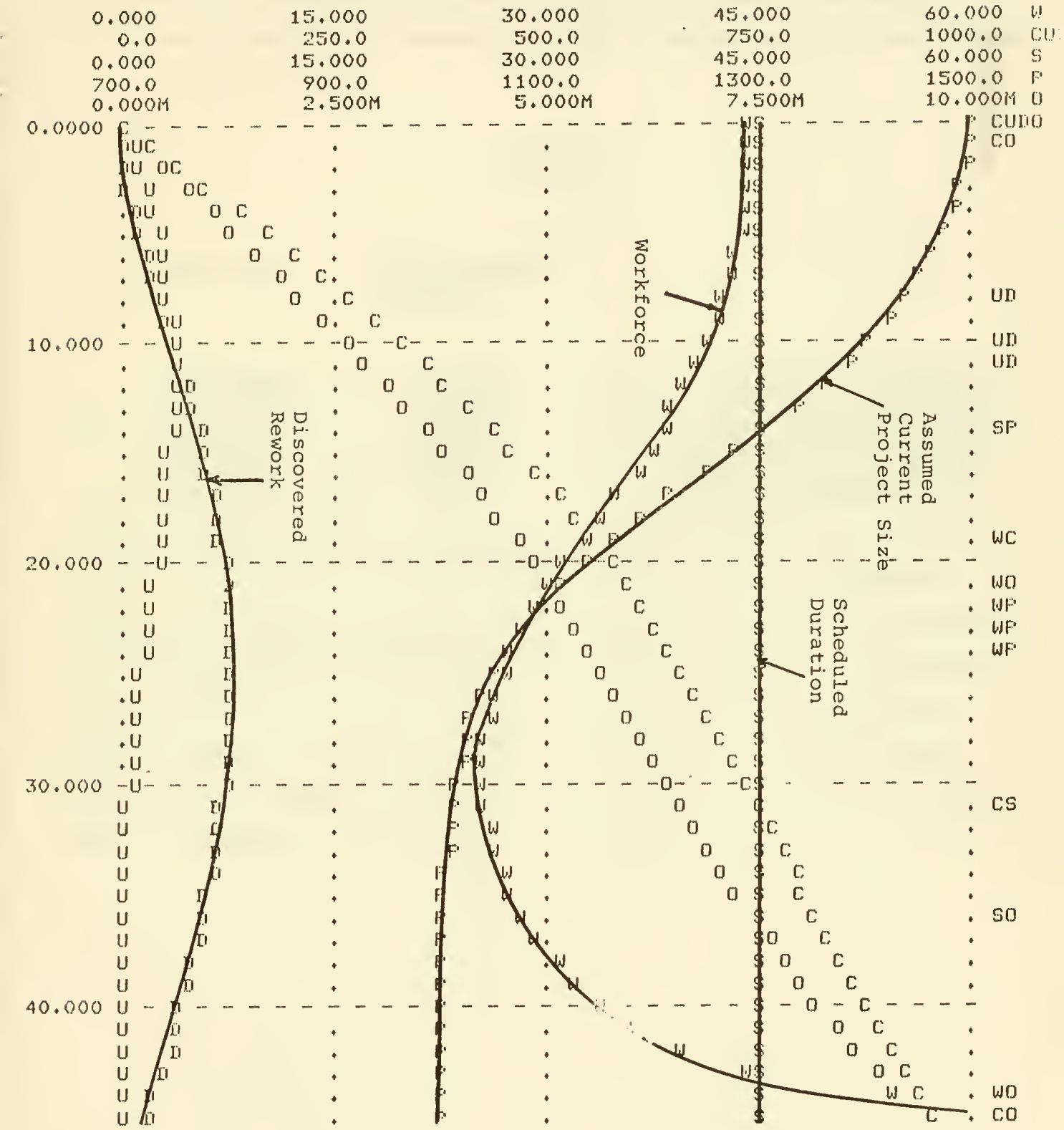


Figure (10)

On the other hand, managerial actions and policies seem to be the factor that most influences a software project's total duration time.

	Completion-Time (months)	Cost ($)
"Flawless" Project	38.25	7,817,800
Introducing Rework	41.50	9,390,700
Introducing Turnover	46.00	9,636,000
Introducing Est. Error	51.50	9,394,600
Fixed-Workforce Policy	36.00	9,567,800
Fixed-Schedule Policy	45.00	9,505,300

Table (1)

SUMMARY

In this paper our concern has been with the management of software development projects. In Section (I) we argued for the attractiveness of using computer modeling techniques in the study and design of complex social systems. In particular, we advocated the use of the modeling, simulation, and analysis techniques of System Dynamics. In Section (II) we, then, presented a System Dynamics model we developed for software project management. The model's three major subsystems: the human resource management subsystem, the planning and control subsystem, and the software production subsystem, were discussed in some detail. Finally, in Section (III), the dynamic behavior of the model was analyzed using a series of simulation runs.

BIBLIOGRAPHY

1. Boehm, B. W. Software Engineering Economics. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981.

2. Forrester, J. W. "Counterintuitive Behavior of Social Systems." Technology Review, January, 1971.

3. Richardson, G. P. and Pugh III, A. L. Introduction to System Dynamics Modeling with Dynamo. Cambridge, Massachusetts: The MIT Press, 1981

4. Roberts, E. B., ed. Managerial Applications of System Dynamics. Cambridge, Massachusetts: The MIT Press, 1981

5. Weil, H. B. "Industrial Dynamics and Management Information Systems." In (Roberts, 1981)

BASEMENT
Date Due

SEP 3 0 1983

AUG 05

FEB 1990

OCT. 20 1995

APR.

FEB 2 8 1999

Lib-26-67

HD28.M414 no.1333- 82
Abdel-Hamid, T/A model of software pro
745064 D*BKS 00136965
3 9080 002 047 584